# Practical solutions for a dock assignment problem with trailer transportation

Lotte Berghman *[a], Roel Leus[b]

[a] *Université de Toulouse – Toulouse Business School*
*20 BD Lascrosses, BP 7010, 31068 Toulouse Cedex 7, France*
*l.berghman@tbs-education.fr*
[b] *ORSTAT, KU Leuven*
*Naamsestraat 69, 3000 Leuven, Belgium*
*Roel.Leus@kuleuven.be*

**Abstract**

We study a distribution warehouse in which trailers need to be assigned to docks for loading or unloading. A parking lot is used as a buffer zone and transportation between the parking lot and the docks is performed by auxiliary resources called terminal tractors. Each incoming trailer has a known arrival time and each outgoing trailer a desired departure time. The primary objective is to produce a docking schedule such that the weighted sum of the number of late outgoing trailers and the tardiness of these trailers is minimized; the secondary objective is to minimize the weighted completion time of all trailers, both incoming and outgoing. The purpose of this paper is to produce high-quality solutions to large instances that are comparable to a real-life case. This will oblige us to abandon the guarantee of always finding an optimal solution, and we will instead look into a number of sub-optimal procedures. We implement four different methods: a mathematical formulation that can be solved using an IP solver, a branch-and-bound algorithm, a beam search procedure and a tabu search method. Lagrangian relaxation is embedded in the algorithms for computing lower bounds. The different solution frameworks are compared via extensive computational experiments.

*Keywords:* dock assignment, multicriteria scheduling, branch and bound, beam search, Lagrangian relaxation, tabu search

## 1. Introduction

We study a distribution warehouse with several docks, where incoming trailers are unloaded after they arrive and where outgoing trailers are loaded before they leave. Each dock can be occupied by at most one trailer at any moment in time. The site also contains a parking lot, which serves as a buffer area where trailers are temporarily parked. We distinguish between three types of trailers. First of all, we have the *coupled* trailers. These trailers arrive at the parking lot at a known arrival time (a *release date*) and are brought to the dock

*Corresponding author. Tel +33 5 61 29 47 54.

by the trucker, who waits until the load or unload activity is completed to take the trailer away. A desired latest departure time (a *due date*) is specified for each of these trailers to avoid having truckers wait for excessive time at the plant. A second type of trailers are the uncoupled *incoming* trailers that are to be unloaded. These trailers also have a known arrival time (a release date) but no restrictive due date. The third set of trailers are the uncoupled *outgoing* trailers to be loaded, which are available at the parking area from the outset. These trailers have a due date, since they need to be transported to clients after being loaded. Uncoupled trailers (both incoming and outgoing) are dropped off by a trucker at the parking lot and afterwards transferred to a dock by one of the *terminal tractors*, which are tractors designed for use in ports, terminals and heavy industry. After unloading or loading at the dock, the uncoupled trailer is moved back to the parking lot by a tractor, where it will be picked up by a trucker later on.

The described dock assignment problem is modeled after a case encountered at Toyota Parts Center Europe (TPCE), a Toyota warehouse in Diest, Belgium. The stated assumptions closely adhere to this practical situation. The purpose of this paper is to produce a baseline schedule for the entire next day. Such a baseline provides a convenient check whether the available capacity is sufficient for the upcoming operations, and it serves as a clear guideline for prioritization of those operations. Our task was to develop an automated procedure for building this schedule, which was at the time of our contacts being built mainly by hand. During the execution of this baseline schedule, TPCE obviously receives updated information about the planned arrival time of trucks (for instance based on GPS-tracking information), and some trucks without tracking information will also inevitably arrive somewhat earlier or later than planned. Based on this new information that gradually becomes available, the baseline schedule is manually adjusted in real time.

In an earlier paper (Berghman et al., 2014), we have explored the possibility of finding optimal solutions by means of integer (linear) programming (IP). In that study, the due dates were treated as strict deadlines. After discussion with the management of the TPCE site, however, it turned out that these latest departure times were better modeled as due dates: the existence of a feasible schedule meeting all due dates is not guaranteed, but satisfying them is our primary objective. Minimization of the waiting times of the trailers is the secondary objective. We ambition to produce high-quality solutions to realistic instances. Our results in Section 4 will indicate that large instances cannot be solved to guaranteed optimality within reasonable running times, and we will therefore resort to the development of heuristic procedures. The contributions of this text are fourfold: (1) we cast the practical problem setting into a hierarchical bi-objective optimization problem; (2) we present an IP formulation for this problem; (3) we propose different heuristic algorithms; and (4) we investigate how Lagrangian relaxation can lead to lower bounds. Our computational experiments will show that the best solutions are obtained by a hybrid algorithm that combines recovery beam search and tabu search.

On a practical note, we need to mention that in spite of the promising computational performance that we are able to report, a practical implementation of our algorithms at the Toyota site seems rather unlikely: our (very enthusiastic) direct contact person at TPCE has left the company, and the interest of the remaining team members responsible for logistics in applying scientific methods for planning, is very low. The recent economic crisis and additional company-related downturns have even further distracted the team's interest. We

are convinced, however, that the documented work will be useful to readers confronted with similar planning problems in a practical setting.

The remainder of this article is structured as follows. Section 2 provides a brief literature survey on the related topics of dock scheduling, multicriteria scheduling and flexible flow-shop scheduling. In Section 3, some definitions and a formal problem statement are presented. An IP formulation will be proposed in Section 4. Section 5 explains how schedules will be represented in our solution procedures and Section 6 proposes simple heuristics to provide initial solutions. Subsequently, a branch-and-bound (B&B) algorithm, a beam search algorithm and a tabu search algorithm will be represented in Sections 7, 8 and 9, respectively. An overview of our computational results is given in Section 10 and we round off the article with a summary and some conclusions in Section 11.

## 2. Literature review

In this section, we provide a brief review of the recent relevant work in different fields. First, we describe the literature on dock scheduling including cross docking (Section 2.1). Secondly, the literature on multicriteria scheduling is surveyed in Section 2.2 and finally, a brief overview is included of the literature on flexible flow-shop scheduling (Section 2.3).

### *2.1. Dock scheduling*

The problem presented in this paper is a dock assignment problem: trailers are assigned to docks for a limited period of time for loading or unloading activities. The storage capacity of the warehouse is not restricted and there are no links between incoming and outgoing shipments. All goods stay at least one night in the warehouse, such that a product that is unloaded at day $X$ will be forwarded at day $X + 1$ at the earliest. We are not aware of existing scientific papers with exactly the same setup.

A related setting with trailer scheduling that has received attention in the recent literature is cross docking. According to Yu and Egbelu (2008), "Cross docking is a warehouse management concept in which items delivered to a warehouse by incoming trucks are immediately sorted out, reorganized based on customer demands, routed and loaded into outgoing trucks for delivery to customers without the items being actually held in inventory at the warehouse." The advantages are faster deliveries, lower inventory costs and a reduction of the warehouse space requirement. A comprehensive overview of different variations and the available literature can be found in Boysen and Fliedner (2010) and van Belle et al. (2012). The truck-dock assignment problem examines the scheduling of a set of trailers at docks over time (Miao et al. 2009). The dock assignment problem is similar to the truck-dock assignment problem in cross docking, but in our case there is no explicit restriction on the warehouse capacity and the incoming and outgoing shipments are unrelated, so there are no precedence constraints between different trailers.

### *2.2. Multicriteria scheduling*

When a schedule's quality is evaluated on multiple performance criteria, in most cases there will be no schedule that achieves the optimal value for all criteria simultaneously and a tradeoff needs to be struck, which depends on the preferences of the decision maker. A common approach for dealing with such multicriteria scheduling problems is to aggregate

the different criteria into one composite objective function, a process that is often called *scalarization* or *simultaneous optimization* (Baker and Smith, 2003; Hoogeveen, 2005). If one criterion is dominant, however, the decision maker will prefer to first distinguish the set of all schedules that are optimal with respect to the primary objective and then search within this set of schedules for one that is best for a secondary objective (Pinedo, 2008). This approach is called *hierarchical* or *lexicographic optimization* and is an example of a *non-scalarizing method* (Sarin and Hariharan, 2000; T'kindt et al., 2003).

### *2.3. Flexible flow-shop scheduling*

The dock assignment problem studied in this paper can be modeled as a flexible flow shop. In a flexible flow shop, also called hybrid or multi-processor flow shop, at least one stage consists of parallel machines. The terminal tractors in this paper can be modeled as machines rather than transporters, especially since the time it takes the tractors to convey a trailer between the docks and the parking lot is essentially independent of the distance (see Section 3). In this way, the transportation activities become stages one and three of a flexible flow shop, and the load/unload activities constitute the second stage. In our problem, the same set of identical machines (the tractors) executes both the first and the third stage of the uncoupled trailers, while the second stage of all trailers takes place on another set of identical machines (the docks). None of these machines are needed for the first and the third stage of the coupled trailers. In a slightly different setting, scheduling with multiple resource types has also been studied by Blazewicz et al. (1999).

Linn and Zhang (1999), Vignier et al. (1999) and Ribas et al. (2010) all provide surveys of the flexible flow-shop literature. Most studies deal with two-stage flow shops with parallel machines either in the first or in the second stage, but not in both. Many research articles related to flexible flow-shop scheduling are available, but most of these do not handle unequal ready times. Both approximation (see, e.g., Tang and Xuan 2006; Nichi et al. 2010) and optimal approaches (for instance Kis and Pesch 2005; Haouari et al. 2006) have been published.

A limited number of articles propose solution procedures for flow-shop scheduling with release times. Moursli and Pochet (2000) introduce a B&B algorithm for makespan minimization that produces high-quality results even when it is truncated after a few minutes of computation time. Gupta et al. (2002) generalize well-known heuristic approaches and present constructive algorithms based on job insertion techniques and iterative algorithms based on local search. Paternina-Arboleda et al. (2008) propose a heuristic for makespan minimization that focuses on the identification and exploitation of the bottleneck stage.

A flowshop where a job may return one or more times to a previously visited machine, is called a reentrant flowshop. Although these flowshops are usually operated and scheduled as general job shops (Graves et al., 1983), some dedicated algorithms can be found in literature (see, e.g., Kubiak et al. 1996; Chen et al. 2007; Choi and Kim 2008). Due to the specificity of our dock assignment problem, however, we will develop new models for producing optimal and heuristic solutions.

## 3. Definitions and problem statement

We pointed out in Section 2.3 that three tasks are performed for each trailer (corresponding to three stages in a flexible flow shop): the movement from the parking lot to a

dock, the loading or unloading task and the transportation back to the parking area. The decisions to be made are the timing of each of the three tasks, the choice of the tractor for stages one and three of the uncoupled trailers, and the choice of the dock for stage two of all trailers. The load/unload times may differ between jobs but do not depend on the dock. The transportation activities are modeled as having a constant duration because the time to follow the safety instructions and fulfilling administrative requirements is large compared to the actual transportation time. The parking lot and the gates are located very close to each other, so the driving time only makes up a small part of the total shunting time. In reality, not all trailers will need exactly the same duration for safety and for administration, but 10 minutes will suffice to cover these activities for all trailers. This value thus serves as a robust estimate of the processing time of these tasks in the baseline schedule to be developed, and the durations are modeled as a constant. Since the durations of the loading and unloading activities are all expressed as multiples of 10 minutes, we decide to work with time periods of this length. Consequently, the transportation activities have a duration of one time unit.

The set $J$ contains all jobs (trailers), with $|J| = n$, and $T$ is the set of all tasks. Each job $j \in J$ is a vector $(t_1, t_2, t_3)$ of three tasks, one at each stage (the first component is the task in the first stage, etc.). Set $T$ can be partitioned as follows: $T = T^1 \cup T^2 \cup T^3$, with $T^i$ the set of all tasks of stage $i$ ($i = 1, 2, 3$). A second partition is $T = T_C \cup T_U \cup T_L$, where the set $T_C$ contains all tasks related to trailers that will remain coupled to the truck, $T_U$ gathers all tasks related to an uncoupled trailer that has to be unloaded, and $T_L$ contains all tasks pertaining to an uncoupled trailer that has to be loaded. Each task $t \in T^1$ has a ready time $r_t$; for $t \in T_L$, $r_t = 0$. With each task $t \in T^2$ we associate a processing time $p_t$, denoting the time to load or unload the trailer. Each third-stage loading task $t \in T_L \cap T^3$ has a due date $d_t$, which is based on the agreed arrival time at the customer. Each third-stage coupled task $t \in T_C \cap T^3$ also has a due date, which creates a time window for the coupled trailers. This window is meant to restrict the trucker's time at the site. Each of the tasks $t \in (T_U \cap T^2) \cup (T_L \cap T^3) \cup (T_C \cap T^3)$ also has a weight $w_t$, representing the importance of the job. The weights are chosen based on the product type and on the transportation mode towards the client. At TPCE, a higher weight applies for outbound trailers that need to be loaded on a train or a boat, for instance. The resources in the second stage are $m < n$ identical docks (also called 'gates'), and $\tau < m$ identical terminal tractors execute both the first and the third stage of the uncoupled trailers. The trucks that transport coupled trailers are not explicitly modeled as resources because they are not shared among the different trailers. Each machine (either a dock or a tractor) can process at most one task at a time and preemption of a task is not allowed.

During the transportation stages one and three, the dock is also considered to be occupied, mainly for safety reasons. As a consequence, a stage-two task always starts immediately at the end of the corresponding stage-one task: the selected dock is always free. A stage-three task of a coupled trailer will also start directly at the end of its stage-two task. A stage-three task of an uncoupled trailer, on the other hand, can start as soon as the corresponding stage-two task is finished, but will regularly be delayed because of unavailability of tractors. This leads to the phenomenon of *blocking*: as long as the stage-three task for a trailer is not executed, the assigned dock remains occupied although stage two may already be completed. Consequently, also the 'dock'-resources are not exclusively tied to only one stage. An example

Table 1: Data for the example instance. Type 'C' are coupled jobs, type 'U' are unload jobs and type 'L' are load jobs. All parameters (weight, ready time, ...) pertain to the appropriate tasks of each job.

| job | weight | ready time | processing time | due date | type |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 12 | 17 | C |
| 2 | 3 | 3 | 12 | | U |
| 3 | 3 | 1 | 14 | | U |
| 4 | 2 | 0 | 10 | 15 | L |
| 5 | 1 | 0 | 11 | 25 | L |

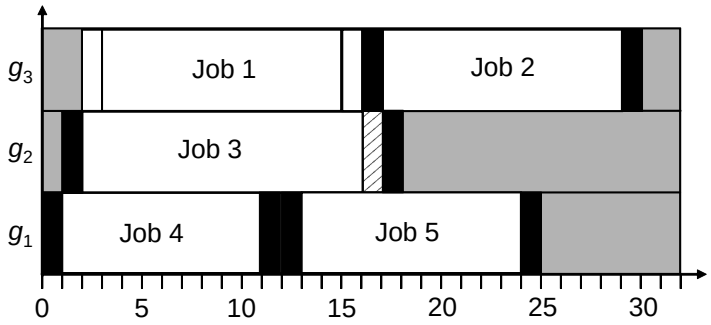


Figure 1: A feasible schedule for $m = 3$ and $\tau = 1$.

instance with five trailers ($n = 5$) is described in Table 1. A feasible schedule with one tractor ($\tau = 1$) and three docks ($m = 3$) is depicted in Figure 1 ('$g_i$' stands for gate/dock $i$). Each job is visually represented by three blocks, one for each stage. White blocks represent occupation of the dock and black blocks represent occupation of both the tractor and the dock. Note that the tractor is not assigned to the transportation of job one, since it is a coupled job. The hatched block represents blocking: trailer three remains at dock two although unloading is already finished because the terminal tractor is occupied at dock 3 during time period 17.

A final constraint is that the schedule length cannot exceed $H_{\max}$, the length of the time horizon. Unless otherwise mentioned, we impose $H_{\max} = 120$, representing a working day of 20 hours (each time unit corresponds to 10 minutes). With this common deadline, verifying the existence of a feasible schedule becomes NP-complete (since the decision variant of $P||C_{\max}$ is NP-complete, by straightforward reduction from 3-PARTITION; see Garey and Johnson (1978)).

Informally, our goal is to have all outgoing and coupled trailers ready for transportation by their due date and also to perform all tasks as quickly as possible. The first objective of respecting due dates is far more important than the overall desire of 'early' processing, and we opt for lexicographic optimization. The primary objective is to live up to the due dates as well as possible. The minimization of the weighted sum of completion times is our secondary objective, where for incoming jobs the completion time of stage two is important, while for coupled and outgoing jobs we focus on the completion time of stage three. In Berghman

et al. (2014), optimization only of the second objective was studied, while due-date violation was forbidden (due dates were deadlines). In practice, however, it turns out that the due dates are tight and a feasible plan without violation might not exist. After discussions with the site management, we have opted for modeling the primary objective by means of two components: we minimize the weighted sum of the number of late coupled and outgoing trailers and the tardiness of those trailers. Formally, our objectives are

$$\min \quad z_1 = \sum_{t \in (T_C \cap T^3) \cup (T_L \cap T^3)} \theta_t + \alpha \delta_t$$

and

$$\min \quad z_2 = \sum_{t \in (T_C \cap T^3) \cup (T_L \cap T^3) \cup (T_U \cap T^2)} w_t C_t,$$

where $C_t$ is the completion time of task $t$, $\theta_t = \max\{C_t - d_t; 0\}$ is the tardiness of $t$ and $\delta_t$ is a binary indicator equal to 1 if $\theta_t > 0$, and 0 otherwise.

The composite objective $z_1$ was chosen because lower tardiness is better for a trailer, but we may prefer having one trailer late by two time periods over having two trailers late each by one time period, because each tardiness occurrence will give rise to communication with the client, loss of time and a possible loss in revenues and/or reputation. For this reason, we also incorporate the number of late trailers. The value $\alpha \geq 0$ serves as a scaling parameter between these two client-oriented performance measures. Consequently, objective $z_1$ is an extension of the total tardiness objective and incorporates a fixed as well as a proportional tardiness cost for every tardy job. It has a very practical sense: if a trailer is tardy then the warehouse incurs a penalty that increases proportionally over time. The weighted completion-time objective $z_2$ is also convenient for our setting: all the incoming goods will be stored in the warehouse as early as possible and all outgoing trailers will be in the parking zone as quickly as possible, ready for transportation towards the client. Additionally, in case of coupled trailers, we also reduce the trucker's stay on site. Optimization of $z_1$ has priority over $z_2$: improving $z_1$ is crucial, even if it causes a worsening in $z_2$ (hierarchical optimization).

Unless otherwise mentioned, the different solution methods tested below will each be allotted one hour of computation time for each instance. This time limit was imposed by the TPCE management. Letting the computations run overnight was not an option because the management of the site found it important to have a full schedule ready by the end of the previous working day in order to cross-check various external contacts with clients and suppliers.

## 4. Integer programming

Various IP formulations were explored in Berghman et al. (2014) for minimizing $z_2$ with deadlines. A time-indexed formulation was consistently found to be the most efficient. For this reason, we will adapt that formulation to our multicriteria setting with due dates. Let *(time) period* $u$ be the time interval $[u-1, u[$. For all tasks $t \in T$ and for all time periods $u \in H_t$, we define variable

$$x_{tu} = \begin{cases} 1 & \text{if task } t \text{ starts in period } u, \\ 0 & \text{otherwise,} \end{cases}$$

with $H_t$ the time window for $t \in T$. Specifically, $H_t = \{r_t + 1, \ldots, H_{\max} - p_t - 1\}$ if $t \in T^1$, $H_t = \{r_t + 2, \ldots, H_{\max} - p_t\}$ if $t \in T^2$ and $H_t = \{r_t + 2 + p_t, \ldots, H_{\max}\}$ if $t \in T^3$ (each task has to be finished by $H_{\max}$, so we choose the latest starting time for each task as $H_{\max}$ minus the duration of the considered task and of its successors). Furthermore, for all tasks $t \in (T_C \cap T^3) \cup (T_L \cap T^3)$, consider $\theta_t$ and $\delta_t$ as defined in Section 3. A linear formulation with these variables and with two objectives is:

$$\min \quad z_1 = \sum_{t \in (T_C \cap T^3) \cup (T_L \cap T^3)} \theta_t + \alpha \delta_t \tag{1}$$

$$\min \quad z_2 = \sum_{t \in T_U \cap T^2} w_t \left( \left( \sum_{u \in H_t} u x_{tu} \right) - 1 + p_t \right) + \sum_{t \in (T_C \cap T^3) \cup (T_L \cap T^3)} w_t \left( \sum_{u \in H_t} u x_{tu} \right) \tag{2}$$

subject to

$$\sum_{u \in H_t} x_{tu} = 1 \qquad \forall t \in T \tag{3}$$

$$\sum_{u \in H_t} u x_{tu} - d_t - \delta_t H_{\max} \leq 0 \qquad \forall t \in (T_C \cap T^3) \cup (T_L \cap T^3) \tag{4}$$

$$\sum_{u \in H_t} u x_{tu} - d_t - \theta_t \leq 0 \qquad \forall t \in (T_C \cap T^3) \cup (T_L \cap T^3) \tag{5}$$

$$\sum_{(t_1,t_2,t_3) \in J} \left( x_{t_1 u} + x_{t_3 u} + \sum_{v \leq u} (x_{t_2 v} - x_{t_3 v}) \right) \leq m \qquad \forall u \in \{1, \ldots, H_{\max}\} \tag{6}$$

$$\sum_{(t_1,t_2,t_3) \in J_U \cap J_L} (x_{t_1 u} + x_{t_3 u}) \leq \tau \qquad \forall u \in \{1, \ldots, H_{\max}\} \tag{7}$$

$$x_{t_1 u} - x_{t_2, u+1} = 0 \qquad \forall (t_1, t_2, t_3) \in J; \forall u \in H_{t_1} \tag{8}$$

$$\sum_{v=1}^{u} x_{t_3 v} - \sum_{v=1}^{u - p_{t_2}} x_{t_2 v} \leq 0 \qquad \forall (t_1, t_2, t_3) \in J; \forall u \in \{1, \ldots, H_{\max}\} \tag{9}$$

$$x_{tu} \in \{0, 1\} \qquad \forall t \in T; \forall u \in H_t \tag{10}$$

$$\delta_t \in \{0, 1\} \qquad \forall t \in (T_C \cap T^3) \cup (T_L \cap T^3) \tag{11}$$

$$\theta_t \geq 0 \qquad \forall t \in (T_C \cap T^3) \cup (T_L \cap T^3) \tag{12}$$

Objective function (1) minimizes the weighted sum of the number of late coupled trailers and late uncoupled outgoing trailers, and the tardiness of those trailers. Objective (2) minimizes the weighted completion time of the stage-two tasks of the incoming trailers and the stage-three tasks of the coupled and outgoing trailers. Constraints (3) require each task to be processed exactly once. Constraints (4) establish whether a job is late or not and constraints (5) measure the tardiness. Constraints (6) ensure that in each time period, at most $m$ docks are occupied. Constraints (7) enforce the capacity of the terminal tractors. Constraints (8) and (9) implement the precedence constraints between the three stages. We observe that a stage-two task can always begin immediately after the corresponding stage-one task has been completed. For reasons of clarity, the model above includes all variables relating

to the three stages, but for actual computations the stage-two variables are eliminated via substitution according to (8). Optimization proceeds in two steps: first, objective (1) is optimized subject to constraints (3)–(12), leading to objective value $z^*$. Subsequently, the constraint $z_1 \leq z^*$ is added and then objective (2) is optimized. The obtained objective value will be denoted as $z_2(z^*)$. Alternatively, optimization can also proceed in one single step by minimizing $Mz_1 + z_2$, with $M = nw_{\max}H_{\max}$ an upper bound on $z_2$, where $w_{\max}$ is the maximum weight over all trailers.

All algorithms in this article were encoded in C using the Microsoft Visual Studio programming environment, and executed on a Lenovo Thinkpad X220i with an Intel Core i3 2.3-GHz processor and 2 GB RAM, equipped with Windows 7. CPLEX version 12.4 is used to solve the IP and LP models. In all implementations, we choose $\alpha = 1$ unless otherwise mentioned. Based on Sadykov and Wolsey (2006) and Berghman et al. (2014) and in line with the current situation in the case studied, instances were created in the following way: 25% of the trailers remains coupled, 30% is uncoupled and has to be unloaded, and the remaining 45% are uncoupled trailers to be loaded. The ready times for the coupled and the incoming trailers are integers randomly selected out of $[0, 64]$ and the weights of all trailers are randomly selected out of $\{1, 2, 3\}$ (each value has equal probability). The processing times $p_t = 1 + X$ with $X$ binomially distributed with 16 trials and a probability of 0.5. The due dates for the coupled trailers are obtained as $d_t = r_t + p_t + 18$, while the due dates for the outgoing jobs are calculated in the following way: $d_j = \max\{d'_j, r_j + \max_{k\in J}\{p_k\}\}$ with $d'_j \in [\beta - 10, \beta + 10]$ and $\beta = \frac{\sum_{j\in J} p_j * 0.5}{m}$. For the case, the number of tractors is always very low compared to the number of gates. At TPCE, the tractors are actually rented on a monthly basis, and so the number $\tau$ is, to a certain extent, a decision variable (albeit not for our daily operational planning horizon), but financial considerations do not allow to employ a similar number of tractors as there are gates.

Table 2 displays the objective values and the computation times for medium-sized instances for both the IP formulations and for their LP relaxations; the latter yield lower bounds $LB_1$ on $z_1$ and $LB_2(z_1)$ on $z_2(z_1)$. We have evaluated the performance of the solver with parameter settings that emphasize feasibility and focus less on proof of optimality (ILOG 2008); with this new setting, however, CPLEX was not able to find a feasible solution for more instances. Moreover, the objective values found were sometimes worse than with the (initial) balanced setting (equal emphasis on feasibility and proof of optimality). We therefore report the results for the latter setting. We include the lower bounds because they may be useful for later sections. Here and below, '$\text{time}_i$' represents the time spent by the relevant procedure on optimizing objective $z_i$ and 'time' represents the time consumed by the one-step procedure. For most instances, the single-step optimization is significantly faster than the two steps separately, although it does not find a guaranteed optimal solution within the time limit for one of the instances (entry 'no opt sol'). For larger and more realistic instances with more trailers per dock (up to 48 gates, 480 trailers and over 1000 operations (tasks)), CPLEX is no longer able to produce optimal solutions: see Table 3. For most of those cases, CPLEX is even unable to find a feasible solution within one hour of computation time, or is aborted because of memory problems.

Table 2: Computational results for medium-sized instances for the IP formulations and the LP relaxations.

| $m$ | $n$ | $\tau$ | IP | | | | | LP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $z_1$ | $z_2(z_1)$ | time$_1$ (s) | time$_2$ (s) | time (s) | $LB_1$ | $LB_2(z_1)$ | time$_1$ (s) | time$_2$ (s) |
| 20 | 80 | 2 | 190 | 6167 | 36.38 | 40.74 | 32.47 | 158.31 | 5884.01 | 8.24 | 15.08 |
| 20 | 80 | 3 | 36 | 5694 | 31.31 | 17.97 | 41.18 | 2.84 | 5622.54 | 7.82 | 9.46 |
| 20 | 90 | 2 | 227 | 6570 | 46.75 | 50.29 | 44.58 | 187.55 | 6218.63 | 12.72 | 29.43 |
| 20 | 90 | 3 | 45 | 5883 | 33.14 | 29.79 | 44.40 | 0.00 | 5815.20 | 13.27 | 50.52 |
| 20 | 100 | 2 | 256 | 8187 | 70.61 | 45.05 | 42.00 | 206.20 | 7780.08 | 11.15 | 26.93 |
| 20 | 100 | 3 | 10 | 7233 | 32.07 | 18.10 | 43.61 | 0.00 | 6960.89 | 8.60 | 33.18 |
| 24 | 96 | 2 | 830 | 7815 | 44.71 | 47.55 | 55.26 | 785.49 | 7494.80 | 12.62 | 16.76 |
| 24 | 96 | 3 | 722 | 6704 | 29.52 | 37.05 | 49.06 | 711.88 | 6611.82 | 6.71 | 11.66 |
| 24 | 108 | 2 | 464 | 9953 | 58.81 | 55.67 | 48.36 | 425.01 | 9437.84 | 12.17 | 34.12 |
| 24 | 108 | 3 | 98 | 8414 | 41.66 | 55.03 | 50.91 | 21.87 | 7954.48 | 13.45 | 24.44 |
| 24 | 120 | 2 | 522 | 11371 | 85.15 | 75.61 | 63.92 | 482.99 | 10828.57 | 15.34 | 57.87 |
| 24 | 120 | 3 | 73 | 9119 | 45.21 | 114.12 | 60.08 | 0.00 | 8791.87 | 13.97 | 38.44 |
| 28 | 112 | 2 | 660 | 8987 | 57.72 | 85.72 | no opt sol | 629.70 | 8728.63 | 11.95 | 40.24 |
| 28 | 112 | 3 | 257 | 7470 | 40.30 | 62.26 | 85.26 | 212.76 | 7202.65 | 8.87 | 22.16 |
| 28 | 126 | 2 | 719 | 12492 | 82.99 | 129.42 | 55.64 | 679.11 | 12024.77 | 14.88 | 52.07 |
| 28 | 126 | 3 | 219 | 10138 | 41.78 | 38.01 | 85.75 | 152.26 | 9792.58 | 14.07 | 36.72 |
| 28 | 140 | 2 | 894 | 14302 | 123.59 | 86.94 | 81.87 | 832.88 | 13515.94 | 27.04 | 102.27 |
| 28 | 140 | 3 | 262 | 11436 | 77.97 | 225.14 | 79.79 | 165.87 | 10664.83 | 17.92 | 44.55 |
| 32 | 128 | 3 | 450 | 9792 | 50.60 | 68.00 | 117.31 | 410.39 | 9339.38 | 11.67 | 25.37 |
| 32 | 128 | 4 | 202 | 8806 | 43.47 | 55.35 | 95.65 | 140.16 | 8550.32 | 10.03 | 18.95 |
| 32 | 144 | 3 | 383 | 11942 | 87.06 | 62.65 | 108.80 | 318.13 | 11255.33 | 16.29 | 45.11 |
| 32 | 144 | 4 | 79 | 10715 | 51.45 | 40.82 | 51.34 | 0.00 | 10113.47 | 14.55 | 52.02 |
| 32 | 160 | 3 | 578 | 13378 | 130.24 | 118.15 | 339.49 | 531.90 | 12741.88 | 19.35 | 70.47 |
| 32 | 160 | 4 | 193 | 11574 | 90.22 | 69.48 | 128.64 | 96.30 | 11102.04 | 19.35 | 70.47 |

## 5. Schedule representation and generation scheme

It turns out (based on the previous section as well as on preliminary results for the exact B&B algorithm proposed in Section 7) that we cannot solve realistic instances to guaranteed optimality within reasonable running times and in the remainder of this text, we therefore resort to the development of heuristic procedures for solving the dock assignment problem. The procedures will be discussed in the following sections and afterwards compared experimentally. In this section, we first explain the schedule representation and the schedule generation scheme that will be used by those procedures.

Similar to most improvement heuristics for scheduling problems, we will not operate directly on a schedule but rather on a *representation* of a schedule that admits an efficient and effective functioning of the algorithm. We opt for an (ordered) task list, which will also be referred to as a 'sequence' or 'permutation'; similar choices have been made in a number of branching algorithms (see, e.g., Baker, 1974; Azizoğlu and Kirca, 1999). The tasks of stage two are not included in the list because they always start immediately after the corresponding stage-one task. The third-stage tasks of the coupled trailers are not included either since they start immediately after the corresponding stage-two task. As a consequence, the length of the permutation will be $2n - c$, with $c$ the number of coupled trailers. A number without prime will represent a stage-one task and a number with prime ($'$) a stage-three task (the number is the job index).

A schedule representation is transformed into a schedule by means of a *schedule generation*

Table 3: Computational results for large instances for the IP formulations and the LP relaxations. When $z_1$ is not optimized to completion, an asterisk '*' indicates that CPLEX is unable to optimize the linear-programming relaxation of the secondary objective: an integer solution is needed to provide a primary objective value. For some instances, CPLEX does not find any feasible solution within one hour or encounters memory problems ('no feas sol').

| $m$ | $n$ | $\tau$ | IP | | | | | LP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $z_1$ | $z_2(z_1)$ | time$_1$ (s) | time$_2$ (s) | time (s) | $LB_1$ | $LB_2(z_1)$ | time$_1$ (s) | time$_2$ (s) |
| 36 | 216 | 4 | 489 | 19775 | 210.48 | 178.59 | 230.73 | 312.58 | 18705.58 | 54.57 | 90.36 |
| 36 | 216 | 5 | 118 | 18120 | 199.38 | 669.78 | 3389.68 | 0.00 | 17419.14 | 47.22 | 93.24 |
| 36 | 288 | 5 | | | no feas sol | | | 0.00 | * | 106.70 | * |
| 36 | 288 | 6 | | | no feas sol | | | 0.00 | * | 84.70 | * |
| 36 | 360 | 6 | | | no feas sol | | | 0.00 | * | 223.38 | * |
| 36 | 360 | 7 | | | no feas sol | | | 0.00 | * | 181.06 | * |
| 40 | 240 | 4 | 800 | 21584 | 207.83 | 144.31 | 149.56 | 618.11 | 20267.06 | 61.87 | 75.74 |
| 40 | 240 | 5 | 298 | 19275 | 236.22 | 997.37 | 206.86 | 30.25 | 18173.69 | 62.41 | 96.70 |
| 40 | 320 | 5 | | | no feas sol | | | 0.00 | * | 133.18 | * |
| 40 | 320 | 6 | | | no feas sol | | | 0.00 | * | 110.86 | * |
| 40 | 400 | 6 | | | no feas sol | | | 0.00 | * | 319.38 | * |
| 40 | 400 | 7 | | | no feas sol | | | 0.00 | * | 276.75 | * |
| 44 | 264 | 5 | 541 | 23047 | 195.74 | 863.18 | 419.39 | 263.68 | 21454.51 | 71.11 | 110.75 |
| 44 | 264 | 6 | 189 | 21266 | 232.98 | 367.53 | 707.19 | 0.00 | 20220.70 | 71.62 | 109.14 |
| 44 | 352 | 6 | | | no feas sol | | | 0.00 | * | 139.93 | * |
| 44 | 352 | 7 | | | no feas sol | | | 0.00 | * | 124.62 | * |
| 44 | 440 | 7 | | | no feas sol | | | 0.00 | * | 438.35 | * |
| 44 | 440 | 8 | | | no feas sol | | | 0.00 | * | 405.95 | * |
| 48 | 288 | 5 | 816 | 26483 | 200.30 | 224.37 | no opt sol | 585.84 | 24887.89 | 73.00 | 121.91 |
| 48 | 288 | 6 | 346 | 24099 | 144.70 | 168.83 | 232.96 | 18.15 | 22871.72 | 80.68 | 130.05 |
| 48 | 384 | 6 | 1249 | 38701 | 346.05 | no opt sol | 1314.69 | 500.13 | 36139.38 | 157.49 | 214.34 |
| 48 | 384 | 7 | | | no feas sol | | | 312.58 | * | 52.74 | * |
| 48 | 480 | 7 | | | no feas sol | | | 0.00 | * | 43.90 | * |
| 48 | 480 | 8 | | | no feas sol | | | 0.00 | * | 104.53 | * |

*scheme* (for details, see Kolisch, 1996). We implement a so-called *serial* generation scheme, which iteratively selects the next task in the list and schedules it as early as possible, taking ready times and capacity constraints into account. Only sequences that respect the intra-job precedence constraints will be considered: for each uncoupled trailer, the stage-one task has to precede the stage-three task. The sequence $(4, 3, 1, 2, 4', 5, 3', 5', 2')$, for instance, can be transformed into the schedule in Figure 1. Note that this is not the only sequence leading to this schedule. It can be shown (e.g. Kolisch, 1996) that with a regular objective function (i.e., non-decreasing with task completion times), which is the case both for $z_1$ and for $z_2$, at least one sequence is mapped to an optimal schedule by the serial generation scheme.

Scheduling an uncoupled stage-one task blocks the considered gate from the task's completion until the end of the time horizon; scheduling the corresponding stage-three task makes the gate available again from its ending time onwards. This blocking phenomenon may hamper a straightforward transformation of a task list into a feasible schedule. An illustration is provided in Figure 2, where all docks are blocked after the stage-one tasks

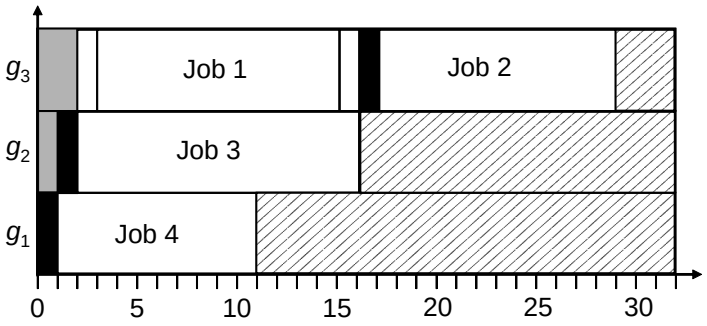


Figure 2: The partial schedule for the permutation $(4, 3, 1, 2, 5, 4', 3', 5', 2')$, after which the generation scheme breaks down.

of trailers 4, 3 and 2 are scheduled because the corresponding stage-three tasks are not yet scheduled. The next task in the permutation is the stage-one task of trailer 5, for which there is no free dock.

We call a permutation *valid* if the generation scheme finds a free dock at each iteration in which a stage-one task is planned, so that the capacity constraints are always respected. A valid permutation does not always generate a feasible schedule, because the overall deadline $H_{\max}$ may still be violated. Let $\gamma(t)$ be the number of stage-one tasks related to uncoupled trailers from the start of the permutation up to, but not including, position $t$; let $\omega(t)$ be the number of stage-three tasks from the start of the permutation up to, but not including, position $t$; and define $\nu(t) = \gamma(t) - \omega(t)$, with $t \in \{2, \ldots, 2n - c\}$ for all three definitions.

**Observation 1** *A permutation is valid if and only if for each position $t \in \{2, \ldots, 2n - c\}$ filled with a stage-one task, $\nu(t) < m$.*

This result is straightforward because when the generation scheme reaches position $t$ in permutation, the number of free docks is exactly $m - \nu(t)$. In the example provided in Figure 2, $\nu(5) = 3 = m$, so the permutation is not valid.

For a given permutation $L$, we define a new permutation $V(L)$ obtained by traversing the permutation from left to right and monitoring $\nu(t)$ for each position $t$. Each time when $\nu(t) = m$, the first stage-three task in the list after position $t$ with stage-one task before position $t$ is inserted at position $t$ and the tasks in between are shifted one position to the right. We observe, based on Observation 1:

**Observation 2** *For any list $L$, the list $V(L)$ is valid.*

As an illustration, the invalid (i.e., not valid) permutation $(4, 3, 1, 2, 5, 4', 3', 5', 2')$ can be transformed into the valid permutation $(4, 3, 1, 2, 4', 5, 3', 5', 2')$ using the above strategy. The resulting schedule was given in Figure 1.

## 6. Starting solutions

In this section, we present a number of simple heuristics that consume only little runtime and that will be used to produce initial solutions for the algorithms that will be proposed in

Table 4: Comparison of different starting solutions.

| | SPT | SWPT | EDD | MSF | R&M |
|---|---|---|---|---|---|
| # infeasible | 2 | 3 | 3 | 4 | 2 |
| $\text{gap}_1$ | 177% | 205% | 59% | 93% | 106% |
| $\text{gap}_2$ | 3% | −3% | 3% | 7% | 2% |

the following sections. We compare the performance of these simple algorithms on a set of test instances.

The first set of algorithms are *static dispatching rules*, which determine the relative position of a task in a sequence by means of an index value for each task that is independent of the other tasks and of the start time of the task. For our problem, the loading and coupled jobs are sequenced first, and the sequence is then completed with the unloading jobs. The loading and unloading jobs are then replaced by both their first-stage and third-stage task (consecutively); the uncoupled jobs are replaced by their first-stage tasks. The SPT heuristic orders jobs in non-decreasing order of their processing time; SWPT sequences the jobs in non-decreasing order of their weighted processing time. The EDD rule sequences the coupled and unloading jobs in non-decreasing order of their due date; all unloading jobs have a non-restrictive due date, and they are sequenced in non-decreasing order of their weighted processing time.

A second set of algorithms are *dynamic dispatching rules*, which are time-dependent (Pinedo, 2008). The MSF (minimum slack first) heuristic schedules the loading and coupled tasks $j$ in order of non-decreasing $(d_j - p_j - t)$, with $t$ the next decision point in the partially constructed schedule. The R&M heuristic (Rachamadugu and Morton, 1981) sequences the coupled and loading tasks $j$ by non-decreasing order of their apparent tardiness cost $ATC$, which is defined as follows: $ATC_j = \frac{1}{p_j} \exp \frac{-d'_j}{2p_{avg}}$, where $d'_j = \max\{0, d_j - p_j - t\}$ and $p_{avg}$ is the average processing time over all jobs. In both dynamic algorithms, the coupled tasks are again ordered by non-decreasing weighted processing time.

We have tested these five algorithms on the 48 instances of Tables 2 and 3. Table 4 shows the number of instances for which the heuristics produce an infeasible schedule, and the average gap for each objective. The gap for the first objective is calculated as follows: $\text{gap}_1 = \frac{z_1 - z^*}{z_1}$, with $z_1$ the objective value of the heuristic and $z^*$ the optimal objective value when CPLEX was able to find it within one hour of computation time. If that is not the case, $z^*$ is the lowest one over all objective values found. Value $\text{gap}_2$ is computed similarly for the second objective as the relative deviation from $z_2(z^*)$, the best value known for $z_2$ subject to the constraint that $z_1 \leq z^*$. We observe that all five the heuristics find feasible solutions for almost all instances. EDD, MSF and R&M provide relatively good upper bounds. The runtimes of all heuristics are very low (fractions of a second). Unless mentioned otherwise, we use the best of the five solutions as a global upper bound in the algorithms that will be described in the following sections.

## 7. Branch and bound

The IP formulation presented in Section 4 is frequently unable to produce *any* feasible solution within the allotted runtime (see Table 3). In this section, we describe a B&B

algorithm for the dock assignment problem. Its running times for obtaining guaranteed optimal solutions have turned out to be excessive for larger instances, regularly even longer than for the IP formulation of Section 4, and we will therefore examine the performance of the algorithm especially in a truncated mode (interrupted after a predetermined time period); the algorithm will be referred to as 'truncated B&B'. An alternative way of exploring the enumeration tree only partially via beam search will be studied in Section 8.

Below, we first comment the branching strategy (Section 7.1) and subsequently provide more details on the dominance rules (7.2), on the bounding procedures (7.3) and on parameterization (7.4).

*7.1. Branching strategy*

The second stage of the dock assignment problem corresponds to a parallel machine scheduling problem. For minimizing the total weighted completion time on parallel machines without ready times, it is a dominant decision to sequence the jobs allocated to a given machine by non-decreasing weighted processing time. Therefore, optimization routines need only be concerned with establishing appropriate job-machine assignments. Azizoğlu and Kirca (1999), for example, propose a B&B algorithm for minimization of the total weighted completion time on parallel machines where at each level of the enumeration tree, a given job is assigned to one of the machines. Procedures for identical parallel machine problems with ready times (see, e.g., Nessah et al., 2008) or which minimize the (weighted) tardiness (see, e.g., Azizoğlu and Kirca, 1998; Shim and Kim, 2007) rely on the fact that an optimal schedule can be constructed by assigning jobs to earliest available machines one by one according to an optimal job priority list (Baker, 1974). Therefore, enumeration schemes fix the elements of a priority list from first to last such that a subproblem corresponds to a partial schedule: a node at the $i^{th}$ level of the tree represents a partial schedule in which the first $i$ positions have been filled, and branching from a node consists in appending an unscheduled job to the end of the partial list.

For the three-stage scheduling problem studied in this article, we will also enumerate priority lists (permutations) by selecting the tasks in the list from the first to the last. A subproblem corresponding to a node at depth $l$ of the search tree is to determine the last $(2n - c - l)$ elements of the permutation, and branching at this node is performed by fixing the $(l + 1)^{th}$ task in the list; the already sequenced tasks are called *fixed tasks*. Nodes for which the fixed tasks do not respect the intra-job precedences or the capacity constraints are immediately discarded. Each level of the tree constitutes a partition of all valid permutations, in which each node represents a subset of permutations with the same initial elements. From this subset, we choose one particular permutation that will be called the *representative* of the node, which is the permutation that most closely resembles the representative of the parent node and respects the branching decision. We refer to the corresponding objective values for the primary and the secondary objective as $rep_1$ and $rep_2$, respectively.

We have also considered other branching strategies for an enumeration algorithm, in an attempt to partially avoid the combinatorial explosion in the enumeration. One possibility would be to branch on resolution options for resource conflicts, in line with, for instance, the B&B procedure of Demeulemeester and Herroelen (1992) for resource-constrained project scheduling, where nodes in the enumeration tree represent resource and precedence-feasible partial schedules. All remaining tasks are then scheduled according to earliest start times,

not taking the resource constraints into account. Branches from a parent node correspond to inclusion-minimal sets of tasks, the delay of which resolves the next resource conflict at the parent node (so-called 'minimal delay alternatives'). Kolisch et al. (1995) find that the performance of this branching strategy is strongly dependent on the parametric characterization of the problem instances; in particular, the less dense the network, the higher the average CPU time. For the problem under study in this paper, the implied precedence network is quite sparse: there is a lot of parallelism due to the flexible flow-shop layout. Consequently, an enumeration procedure based on minimal delaying alternatives would risk incurring very high runtimes as well. We have therefore not implemented this alternative enumeration scheme.

*7.2. Dominance rules*

Define $S(i;\pi)$ as the starting time of task $i$ in the schedule $S(\pi)$ that is generated based on permutation $\pi$, and let $\pi(k)$ be the $k^{th}$ task in permutation $\pi$.

**Observation 3** *Given a permutation $\pi$, if there are two tasks $\pi(k) = i$ and $\pi(l) = j$ with $k < l$ and $S(i;\pi) > S(j;\pi)$, then $\pi$ is dominated.*

This dominance result holds because for schedule $S(\pi)$, an alternative permutation $\pi'$ can be set up with the tasks sequenced in non-decreasing starting time such that $S(\pi') = S(\pi)$. A related result pertains to task pairs with identical starting time:

**Observation 4** *Given a permutation $\pi$, if there are two tasks $\pi(k) = i$ and $\pi(l) = j$ with $k < l$, $S(i;\pi) = S(j;\pi)$ and $i > j$, then $\pi$ is dominated.*

Consider a schedule where two jobs start at the same time $t$, namely a task $j_1$ on machine $m_1$ and a task $j_2$ on machine $m_2$. If we interchange the schedules on machines $m_1$ and $m_2$ from time $t$ onwards, the resulting overall schedule will have the same score on both objective functions. As a consequence, under the conditions listed in Observation 4, there exists a permutation $\pi'$ with the same objective values as $\pi$ and in which $i$ and $j$ are sequenced by increasing job index.

*7.3. Bounds*

A feasible solution to constraints (3)–(12) yields an upper bound $UB_1$ to $z_1$ and also an upper bound $UB_2(UB_1)$ to $z_2$ that is conditional on an upper-bound constraint $z_1 \leq UB_1$. For the example of Table 1, the schedule represented by Figure 1 gives an upper bound $UB_1 = 0$ to $z_1$ and an upper bound $UB_2(0) = 216$ to $z_2$. Upper bounds are *global* bounds: they hold for all nodes in the search tree. Obviously, a value $UB_2(z)$ can also serve as $UB_2(z^+)$ with $z < z^+$. A lower bound $LB_1$ for $z_1$ is obtained as the optimal solution for a relaxation; a lower bound $LB_2(UB_1)$ for $z_2$ corresponds to the optimal solution for a relaxed problem with the addition of an upper-bound constraint $z_1 \leq UB_1$. Lower bounds are *local* bounds: they are specific to one node in the search tree and all its children; a lower bound in the root node is a global bound.

*7.3.1. General*

In each node, the partial schedule with the fixed tasks is monitored and a relaxation of the scheduling problem containing the remaining tasks is solved, where the capacities vary over the time periods (so in the IP formulation, for instance, the right-hand side of (6) and (7) can be different for different $u$). A lower bound is then the sum of the exact objective value for the fixed tasks and a bound for the contribution of the other tasks. It holds that $LB_2(z) \geq LB_2(z^+)$ if $z^+ > z$, since the optimization problem for $z^+$ is a relaxation of the problem associated with $z$.

When for a certain node $LB_1 > UB_1$, the node will be pruned. The same holds for a node where $LB_1 = UB_1$ and $LB_2(UB_1) \geq UB_2(UB_1)$. In this way, a node is eligible for further exploration only when $LB_1 < UB_1$ or when $LB_1 = UB_1$ and $LB_2(UB_1) < UB_2(UB_1)$.

Table 3 shows that CPLEX needs excessive runtimes for solving the LP relaxation; a similar problem arises for the other formulations of Berghman et al. (2014). If we relax the number of tractors (more concretely, we set $\tau = m$), a parallel machine scheduling problem results in which only stage two needs to be considered, after extending the processing times by the transportation times. The resulting bound can be computed by a time-indexed parallel machine formulation based on the one of Berghman et al. (2014). If we relax the number of gates (more precisely, we let $m = n$), we can use CPLEX to solve the resulting parallel machine scheduling problem with precedence constraints in which some jobs do not need machines. It turns out that for both relaxed problems, most of the instances are too hard to solve within one hour of computation time.

Another relaxation that is considered for producing lower bounds is Lagrangian relaxation and produces the lower bounds $LB_1^{LR}$ and $LB_2^{LR}(UB_1)$; this method is explained in more detail in Section 7.3.2.

For each relaxation we can produce a permutation, by sequencing the jobs in non-decreasing order of their starting times and then replacing each coupled job by its corresponding stage-one task and each uncoupled job by its corresponding stage-one and stage-three task. The resulting feasible schedule after applying the serial schedule generation scheme yields an upper bound. The idea of transforming a solution to a relaxation into a feasible solution is not very common for 'standard' B&B algorithms, but has been suggested already in a context of Lagrangian relaxation (see Möhring et al. 1999, 2003).

*7.3.2. Lagrangian relaxation*

To calculate $LB_1^{LR}$, the capacity constraints of both resource types (docks and tractors) are relaxed by means of Lagrange multipliers (see, e.g., Fisher, 1981). Additionally, for computing $LB_2^{LR}(UB_1)$, the extra constraint $z_1 \leq UB_1$ is also relaxed. Since the capacity constraints were the only constraints including more than one job, easily solvable independent job-level subproblems are obtained where the multipliers act as prices that regulate the use of the machines. For each task, the optimal starting time strikes a balance between these machine prices and the original objective function, either the number of jobs late and the total tardiness or the weighted completion time. At each iteration, the relaxed problem is solved and the multipliers are updated by means of subgradient optimization. An overview of the complete procedure is given as Algorithm 1. The two stopping criteria consist of an upper limit on the number of iterations and on the running time; in our implementations, these limits are 60 seconds and 1000 iterations in the root node, and 5 seconds and 10 iterations

**Algorithm 1** The Lagrangian algorithm

1: (initialization)
   $LB := -\infty$; initialize the Lagrangian multipliers $\lambda_u$
2: (relaxation)
   $LB^* :=$ objective value of relaxation with multipliers $\lambda_u$; update $LB := \max\{LB, LB^*\}$
3: (iterate)
   if no stopping criterion is met then update $\lambda_u$ and return to Step 2
4: (output)
   return $LB$

for all other search nodes. When the optimization is halted, a relaxed solution is obtained, which may or may not be feasible. The two most important steps in the computation are explained next.

**Relaxation** The subproblem per job entails the choice of starting times for the different tasks so as to minimize the objective value subject to the intra-job precedence constraints (8) and (9). We evaluate each precedence-feasible combination of starting times and select one with the lowest cost. For the uncoupled trailers, only the tasks of stages one and three need to be scheduled explicitly since the stage-two task is started immediately after stage one. For the coupled trailers, it suffices to schedule only stage one explicitly and select a starting time with lowest cost. A natural implementation has a running time that is linear in the number of jobs and quadratic in the length of the planning horizon.

**Updating the Lagrangian multipliers** Subgradient optimization is an iterative procedure that generates new multipliers starting from an initial set in a systematic fashion. This procedure attempts to find values for the multipliers that yield the maximum lower bound. The initial multipliers are of the form $\lambda'_{u0} = \beta\frac{H_{\max}}{m}$ for the docks and $\lambda''_{u0} = \beta\frac{H_{\max}}{\tau}$ for the tractors ($u \in \{1, \ldots, H_{\max}\}$). The subgradients for the relaxed constraints are $SG'_u = \sum_{(t_1,t_2,t_3)\in J}(x_{t_1u} + x_{t_3u}) + \sum_{(t_1,t_2,t_3)\in J}\sum_{v\leq u}(x_{t_2v} - x_{t_3v}) - m_u$ for the docks and $SG''_u = \sum_{(t_1,t_2,t_3)\in J_U\cup J_L}(x_{t_1u} + x_{t_3u}) - \tau_u$ for the tractors, where $m_u$ represents the number of available docks during time period $u$ and $\tau_u$ represents the number of available tractors during $u$. The step size is $w^{-1}$, with $w$ increasing by $X$ every $Y$ iterations. The multipliers are updated as follows in iteration $i$: $\lambda'_{ui} = \max\{0, \lambda'_{u,i-1} + w^{-1}SG'_u\}$ for the docks and $\lambda''_{ui} = \max\{0, \lambda''_{u,i-1} + w^{-1}SG''_u\}$ for the tractors ($i = 1, 2, \ldots$). The multipliers for the additional constraint on $z_1$ in the optimization of $z_2$ are updated in a similar fashion. Some small experiments were run for parameterization.

### *7.4. Parameterization*

Table 5 shows the results of an experiment run on four large instances[1] in order to evaluate the most convenient algorithmic choices for the truncated B&B algorithm. Four different

[1]More concretely ($m$,$n$,$\tau$) = (36,288,5), (40,320,5), (44,352,6) and (48,384,6).

Table 5: Comparison of different branching choices for the B&B procedure.

| | $\text{gap}_1$ |
|---|---|
| (1) $LB_1^{LR}$ and $LB_2^{LR}$ | 26.47% |
| (2) $LB_1^{LR}$ and $LB_2^{PM}$ | 45.87% |
| (3) $LB_1^{LR}$ and $UB_2$ | 28.50% |
| (4) $UB_1$ and $UB_2$ | 0.00% |

branching choices have been implemented. Each choice uses two 'estimates' for the best value of one objective achievable in each node; each estimate is either an upper or a lower bound. The first value is used to decide which node to explore next, and the second one serves as tie breaker. Superscript '$LR$' refers to Lagrangian relaxation. In choices (1) and (2), the upper bound $UB_1$ on $z_1$ imposed for computing $LB_2(UB_1)$ is the minimum of $rep_1$ and the upper bound on $z_1$ obtained via the $LB_1^{LR}$ computation. The second estimate in choice (3) is the minimum of $rep_2$ and the upper bound on $z_2$ associated with $LB_1^{LR}$. The estimates in choice (4) are $rep_1$ and $rep_2$ respectively. In each setting, only the bounds mentioned are actually computed for fathoming. For each branching choice, we report a gap for the primary objective as before ($\text{gap}_1 = \frac{z_1 - z^*}{z_1}$), but now we take $z_1$ as the objective value for the particular branching choice and $z^*$ as the best objective value over all branching choices, (the 'base' setting, corresponding to 0.00% in the table), when each setting is interrupted after 20 minutes. Choice (1), for instance, reached an objective value that was 26.47% higher on average compared to setting (4). Branching choice (4) achieves the best results. Apparently, the upper bounds reflect rather well the quality of the partial solutions and exploring more nodes is better than having more accurate bounds for branching and pruning.

## 8. Beam search

*Beam search* (see, e.g., Bisiani, 1992; Ball, 2011) is a heuristic framework based on a B&B procedure with a breadth-first tree exploration, which provides a structured approach to a partial examination of an enumeration tree. The technique systematically develops a low number of solutions in parallel in an attempt to find good solutions with minimal search effort. At each level of the tree, only the $b$ most promising nodes are retained as nodes to branch from; the parameter $b$ is called the *beam width*. These nodes are pursued in a breadth-first fashion. Clearly, beam search will tend to require substantially less computational effort than standard B&B procedures, at the expense of the loss of guarantee of finding an optimal solution and the inability to recover from 'wrong' decisions. The same branching choices (here: evaluation functions) as for the B&B algorithm are tested. The values $\text{gap}_1$ in Table 6 are computed similarly as in the previous section. The table indicates that the best branching choice is based exclusively on upper bounds. Based on some preliminary experiments, the beam width is set to $b = 5$. We have also implemented and tested an enhanced beam search procedure that allows for more variation in the selection criteria for search nodes in the framework of what is usually called *filtered beam search*, but this did not lead to better results.

The major disadvantage of beam search is that pruning a node, in particular a node leading to an optimal or to nearly optimal solutions, can never be recovered. The *recovering*

Table 6: Comparison of different branching choices for standard beam search.

| | $\text{gap}_1$ |
|---|---|
| (1) $LB_1^{LR}$ and $LB_2^{LR}$ | 30.13% |
| (2) $LB_1^{LR}$ and $LB_2^{PM}$ | 37.69% |
| (3) $LB_1^{LR}$ and $UB_2$ | 30.97% |
| (4) $UB_1$ and $UB_2$ | 0.00% |

Table 7: Comparison of different branching choices for recovery beam search.

| | $\text{gap}_1$ |
|---|---|
| (1) $LB_1^{LR}$ and $LB_2^{LR}$ | 30.65% |
| (2) $LB_1^{LR}$ and $LB_2^{PM}$ | 42.91% |
| (3) $LB_1^{LR}$ and $UB_2$ | 30.01% |
| (4) $UB_1$ and $UB_2$ | 0.00% |

*beam search* method overcomes this issue by introducing a recovering step at each level of the search tree, which searches for improved partial solutions (see, e.g., Della Croce and T'kindt, 2002; Della Croce et al., 2004; Ghirardi and Potts, 2005; Valente and Alves, 2005; Esteve et al., 2006). At each level of the search tree, once the best $b$ nodes and the corresponding best partial solutions are identified, a recovering step is applied to verify whether a current partial solution is dominated by another partial solution at the same level of the tree. If that is the case, the latter solution becomes the new current partial solution. This modification allows to partially recover from previous wrong decisions and can be seen as a local search on the partial solution.

Since there are no obvious problem-specific dominance rules that can be used here, we opt for so-called *pseudo-dominance conditions* (Ghirardi and Potts, 2005): dominance conditions that are not always valid in general, but work in practice in most cases. Concretely, for a node at depth $l$, we try to insert the task at position $l$ at all positions $k \leq l$, we evaluate the objective values $z_1$ and $z_2(z_1)$ for the resulting partial schedules and we retain the best one. Subsequently, the insertion of the task at position $l-1$ is considered similarly at all positions $k \leq l-1$. These steps are iterated up until the task at position $l - l^*$, with $l^*$ a predefined number. We examine this reduced neighborhood because applying all possible interchange operators to the current partial schedule would take too much time. During this process, we ensure that we have $b$ different partial solutions at each level of the tree. After some preliminary results, we set $b = 2$ and $l^* = 40$.

For node evaluation we use a two-stage approach, where a crude evaluation (*filtering phase*) is applied to select a limited number of nodes that will be evaluated more accurately. For the crude evaluation, we apply the following rule: at the first $2u + c$ levels of the search tree, with $u$ the number of uncoupled loading trailers and $c$ the number of coupled trailers, we only branch on tasks related to coupled trailers and uncoupled loading trailers, because these trailers have due dates and thus can influence the primary objective. For accurate evaluation, the same criteria as for the branching choice in Section 7.4 are tested. Table 7 indicates that the best choice coincides with that for the standard beam search.

Table 8: Parameterization of RBRS.

| | | $\text{gap}_1$ |
|---|---|---|
| $\alpha$ | 0.25 | 3.09% |
| | 0.5 | 0.00% |
| | 1 | 2.59% |
| $X$ | 60 | 0.68% |
| | 75 | 0.00% |
| | 90 | 1.47% |

## 9. Tabu search

Tabu search (see, e.g., Glover, 1989) is a metaheuristic procedure that uses local search to iteratively move from a current solution to a solution in the *neighborhood* of this solution, until some stopping criterion is satisfied. Each solution has an associated neighborhood, and each solution in this neighborhood is reached from the initial solution by an operation called a *move*. At each iteration, a predetermined number of neighbor solutions is created and the best one is retained, even if it does not improve the current objective value. In this way, the chance of becoming trapped in local optima that are not globally optimal, is reduced. To prevent the search from cycling and re-visiting the same solutions many times, the (reverse of the) most recent moves are classified as forbidden or *tabu*. In case a tabu move would result in a very promising solution, however, its tabu classification may be overridden. The *aspiration criterion* that implements this condition evaluates the improvement in the objective value.

We have implemented two tabu-search variants. For the first implementation, a move consists in swapping two random tasks in the permutation and, if needed, rendering the obtained permutation valid. At each iteration, the number of solutions generated equals the number $n$ of jobs. When the objective function is not improved for 750 iterations (250 iterations with 10-minutes runtime limit), the procedure is re-initialized with the next best of the solutions produced by the initial heuristics.

The second implementation attempts to select moves more judiciously rather than simply by swapping random tasks. This can be expected to produce better neighbors, but it will take more computation time. We again generate $n$ solutions at each iteration. In $X\%$ of the cases, a tardy job is inserted earlier in the permutation and in $(1-X)\%$, a non-tardy job is put in a later position. We use regret-based random sampling (RBRS, Drexl 1991) to determine the next job to be moved: the probability of selection of job $j$ is $\frac{(T_j+1)^\alpha}{\sum_{i\in T^t}(T_i+1)^\alpha}$ where $T_j$ represents the tardiness of the job $j$ and $T^t$ is the set containing all tardy jobs. The selected job will be placed $5, 10, 15, \ldots$ and 50 places earlier in the permutation. For the jobs $j$ with $T_j = 0$, we apply RBRS with the load/unload durations $p_j$ for selection; the jobs are then placed $5, 10, 15, \ldots$ and 50 positions further in the permutation. In case the objective is not improved for 750 iterations (250 iterations for 10-minutes runtime), we re-initialize with the next best of the initial solutions. Table 8 shows that $\alpha = 0.5$ and $X = 75\%$ produces the best results for RBRS.

## 10. Computational results

In order to experimentally compare the performance of the algorithms proposed in this paper, we have first run all algorithms for 10 minutes of computation time on the medium-sized instances (which already range between 240 and 480 operations), knowing that the largest instances will be allotted one hour of runtime. Table 9 shows the values of $z_1$ and $z_2$ for the best solution found within the time limit. The average objective values are also reported, together with the average gap (deviation from optimal). We observe here that the standard tabu search algorithm (with random swaps) is the best performing one. Among the tree search algorithms, truncated B&B has the lowest average gap and recovery beam search achieves the lowest average objective value. The column labeled 'hybrid' is commented below.

We have tested the dominance rules of Section 7.2 in a complete B&B algorithm (i.e., run to completion) on small examples (six or eight trailers, two gates and one or two tractors) and the results were quite favorable. For six trailers, the number of visited nodes in the search tree decreased from 62 000 to around 2000 and the computation time decreased from 0.3 seconds to 0.03 seconds. For eight trailers, the number of explored nodes went down from 13 000 000 to about 60 000 and the computation time improved from 85 seconds to less than one second. The first dominance rule was responsible for almost all improvements, which is quite logical for these particular instances because we only have two gates. We have then tested the dominance rules as part of a truncated B&B that was interrupted after 10 minutes on our medium-sized instances; see column 'trunc B&B dom' in Table 9. There were no significant improvements, however. We suspect that the main reason for the absence of improvement is the following. If the search tree is not completely explored, then dominance rules cannot be guaranteed to work: if a node is discarded because it is dominated by another one, then it is still not sure that the other node will be explored (see Gacias et al. (2010) for similar observations). We have therefore decided not to use the dominance rules for the large instances.

We now compare on a dataset with eight medium-size and eight large instances for one hour of computation time. Table 10 shows the values of $z_1$ and $z_2$ for the best solution found within the time limit. The average objective values are also reported, together with the average gap (deviation from the best solution identified per instance). Similarly as for Table 9, we conclude that the recovery beam search algorithm performs slightly better than the standard beam search implementation, and so the pseudo-dominance rule and the limited neighborhood exploration do improve the performance. Recovery beam search does not come out as promising for this problem as it did for other problems, however: the extent of the improvement is only minor compared to previous studies in literature (see, for instance, Ghirardi and Potts (2005) for makespan minimization for unrelated parallel machines and Esteve et al. (2006) for single-machine just-in-time scheduling). Tabu search with random swaps is the best performing algorithm of all those listed in Table 10; it is also better than tabu search with 'intelligent' insert operations and achieves an average gap for $z_1$ of 3.68% for these instances.

After closer examination (not in the tables), it turns out that recovery beam search often makes some drastic improvements in the initial solution over the first five to 10 minutes, but then the objective typically flattens out, and no important improvements are usually found

Table 9: Final computational results for the medium-sized instances with 10 minutes runtime.

| $m$ | $n$ | $\tau$ | trunc B&B | | trunc B&B dom | | beam search | | recovery beam | | TS swap | | TS insert | | hybrid | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ |
| 20 | 80 | 2 | 206 | 6080 | 225 | 6126 | 210 | 6118 | 208 | 6074 | 199 | 6052 | 207 | 6090 | 198 | 6116 |
| 20 | 80 | 3 | 81 | 5751 | 81 | 5751 | 69 | 5697 | 64 | 5717 | 42 | 5680 | 47 | 5707 | 41 | 5712 |
| 20 | 90 | 2 | 250 | 6731 | 250 | 6731 | 249 | 6688 | 255 | 6708 | 235 | 6570 | 247 | 6653 | 236 | 6558 |
| 20 | 90 | 3 | 126 | 6180 | 126 | 6180 | 126 | 6180 | 100 | 6090 | 56 | 5933 | 63 | 5920 | 58 | 5870 |
| 20 | 100 | 2 | 279 | 8508 | 279 | 8495 | 279 | 8508 | 274 | 8450 | 263 | 8288 | 265 | 8336 | 263 | 8276 |
| 20 | 100 | 3 | 52 | 7344 | 86 | 7669 | 99 | 7761 | 89 | 7721 | 26 | 7237 | 54 | 7308 | 23 | 7324 |
| 24 | 96 | 2 | 847 | 7884 | 847 | 7884 | 851 | 7910 | 846 | 7861 | 838 | 7721 | 847 | 7731 | 832 | 7777 |
| 24 | 96 | 3 | 730 | 6904 | 730 | 6904 | 730 | 6904 | 725 | 6837 | 722 | 6722 | 722 | 6794 | 722 | 6724 |
| 24 | 108 | 2 | 484 | 10052 | 484 | 10052 | 481 | 10125 | 483 | 10096 | 475 | 10003 | 478 | 10025 | 472 | 9940 |
| 24 | 108 | 3 | 155 | 8457 | 137 | 8394 | 155 | 8457 | 144 | 8333 | 109 | 8358 | 116 | 8369 | 106 | 8388 |
| 24 | 120 | 2 | 564 | 11636 | 559 | 11629 | 564 | 11636 | 559 | 11641 | 536 | 11383 | 536 | 11471 | 534 | 11282 |
| 24 | 120 | 3 | 136 | 9424 | 136 | 9424 | 136 | 9424 | 118 | 9267 | 81 | 9228 | 100 | 9166 | 83 | 9188 |
| 28 | 112 | 2 | 687 | 9432 | 687 | 9432 | 687 | 9525 | 689 | 9548 | 670 | 8904 | 679 | 9203 | 672 | 9218 |
| 28 | 112 | 3 | 299 | 7624 | 299 | 7624 | 299 | 7624 | 295 | 7642 | 267 | 7491 | 273 | 7488 | 266 | 7443 |
| 28 | 126 | 2 | 742 | 12865 | 742 | 12865 | 7400 | 12823 | 740 | 12948 | 726 | 12626 | 734 | 12756 | 730 | 12535 |
| 28 | 126 | 3 | 261 | 10346 | 261 | 10346 | 261 | 10346 | 254 | 10416 | 235 | 10210 | 234 | 10217 | 229 | 10164 |
| 28 | 140 | 2 | 923 | 14847 | 923 | 14847 | 923 | 14847 | 923 | 14847 | 903 | 14472 | 909 | 14600 | 903 | 14443 |
| 28 | 140 | 3 | 313 | 11496 | 313 | 11496 | 315 | 11446 | 308 | 11498 | 289 | 11380 | 294 | 11368 | 282 | 11319 |
| 32 | 128 | 3 | 484 | 9785 | 484 | 9785 | 489 | 9793 | 480 | 9763 | 464 | 9758 | 469 | 9751 | 464 | 9819 |
| 32 | 128 | 4 | 302 | 9096 | 302 | 9096 | 276 | 9042 | 267 | 8946 | 218 | 8728 | 218 | 8823 | 210 | 8765 |
| 32 | 144 | 3 | 401 | 12165 | 401 | 12165 | 401 | 12209 | 397 | 12251 | 393 | 11956 | 399 | 11956 | 392 | 12033 |
| 32 | 144 | 4 | 177 | 10724 | 177 | 10724 | 177 | 10724 | 152 | 10583 | 104 | 10734 | 126 | 10596 | 94 | 10667 |
| 32 | 160 | 3 | 640 | 13508 | 640 | 13508 | 640 | 13508 | 638 | 13530 | 604 | 13334 | 613 | 13401 | 599 | 13313 |
| 32 | 160 | 4 | 281 | 11909 | 281 | 11909 | 281 | 11947 | 261 | 11901 | 218 | 11601 | 227 | 11544 | 210 | 11569 |
| average | | | 393 | | 394 | | 393 | | 386 | | 361 | | 369 | | 359 | |
| gap | | | 51% | | 65% | | 69% | | 58% | | 14% | | 30% | | 11% | |

Table 10: Comparison of the different algorithms on a limited dataset with one hour runtime.

| $m$ | $n$ | $\tau$ | truncated B&B | | beam search | | recovery beam | | TS swap | | TS insert | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ |
| 20 | 80 | 2 | 206 | 6080 | 210 | 6118 | 208 | 6074 | 199 | 6052 | 207 | 6090 |
| 20 | 90 | 2 | 250 | 6731 | 249 | 6688 | 255 | 6708 | 235 | 6570 | 247 | 6653 |
| 24 | 96 | 2 | 847 | 7884 | 851 | 7910 | 846 | 7861 | 838 | 7721 | 847 | 7731 |
| 24 | 108 | 2 | 484 | 10052 | 481 | 10125 | 483 | 10096 | 475 | 10003 | 478 | 10025 |
| 28 | 112 | 2 | 687 | 9432 | 687 | 9525 | 689 | 9548 | 670 | 8904 | 679 | 9203 |
| 28 | 126 | 2 | 742 | 12865 | 740 | 12823 | 740 | 12948 | 726 | 12626 | 734 | 12756 |
| 32 | 128 | 3 | 484 | 9785 | 489 | 9793 | 480 | 9763 | 464 | 9758 | 469 | 9751 |
| 32 | 144 | 3 | 401 | 12165 | 401 | 12209 | 397 | 12251 | 393 | 11956 | 399 | 12054 |
| 36 | 216 | 4 | 550 | 19951 | 550 | 20260 | 555 | 19956 | 513 | 19731 | 530 | 19864 |
| 36 | 288 | 5 | 552 | 29122 | 580 | 29837 | 549 | 29486 | 536 | 28488 | 566 | 29028 |
| 40 | 240 | 4 | 854 | 22059 | 852 | 21956 | 846 | 22037 | 831 | 21504 | 836 | 21950 |
| 40 | 320 | 5 | 634 | 34107 | 633 | 34329 | 600 | 34333 | 573 | 32859 | 588 | 33489 |
| 44 | 264 | 5 | 642 | 23707 | 642 | 23642 | 654 | 23608 | 592 | 22640 | 616 | 22865 |
| 44 | 352 | 6 | 580 | 36068 | 562 | 36152 | 578 | 36087 | 508 | 35074 | 551 | 35670 |
| 48 | 288 | 5 | 858 | 26824 | 871 | 26785 | 887 | 26904 | 857 | 26294 | 868 | 26544 |
| 48 | 384 | 6 | 1454 | 40222 | 1405 | 40857 | 1389 | 41085 | 1447 | 39885 | 1408 | 40740 |
| average | | | 639 | | 638 | | 635 | | 616 | | 626 | |
| gap | | | 8.23% | | 8.31% | | 7.9% | | 3.68% | | 6.17% | |

afterwards. Tabu search, on the other hand, achieves a steady decrease in the objective function with time, even up to the one-hour time limit. This observation motivates us to study the hybridization of the two procedures where the output of the recovery beam search after limited runtime is used as the first starting solution, replacing the best of the five dispatching rules discussed in Section 6, and then the tabu search is run for the remainder of the runtime. The results after 10 minutes of computation time (two minutes recovery beam search and eight minutes tabu search) for the medium-sized instances are presented in the last column (labeled 'hybrid') of Table 9. Clearly, this hybrid procedure is better than the other algorithms, achieving the lowest average objective value, and the lowest average optimality gap of around 11%; excluding the somewhat aberrant instance $(20, 100, 3)$ with a $z_1$-gap of 150% (objective $z_1 = 25$ versus optimum of 10), the average gap reduces to 4.9%. For the large instances, we will allocate five minutes of CPU time to recovery beam search and 55 minutes to tabu search.

Next to the foregoing sequential hybrid, for the large instances we have also implemented an *alternating* hybrid method, where the recovery algorithm is run for five minutes on each starting solution: on the initial one but also on each new starting solution after re-initialization. Table 11 shows a final comparison of the recovery beam search, tabu search and the hybrid algorithms after one hour of computation time for the dataset with all large instances. For some instances, none of the algorithms finds a feasible solution. This, however, need not indicate a global shortcoming of the algorithms: it may well be that a feasible solution simply does not exist – there is no guarantee from the outset that the instances are feasible. For the instances concerned, we therefore subsequently extend the length of the planning horizon $H_{\max}$ to 144 and rerun all the tests. In practical terms, this means that the day will be run with overtime. The sequential hybrid is the best overall, with an average $z_1$-gap of 6.40% for this dataset. For most of the 24 instances considered, this gap is below 4%, but the average is considerably increased by two instances with a gap of over 20%.

Based on these results, we conclude that the main achievement of this study is the fact that we find high-quality feasible solutions – albeit not always optimal solutions – to very large real-life instances consisting of over 1000 operations. Finding such close-to-optimal solutions to the underlying problem is difficult, to start with because merely verifying whether a feasible solution exists, is already NP-complete, and also because there are no obvious (lower) bounds of good quality: LP cannot be computed anymore for the largest instances, and Lagrangian relaxation turns out to be not strong and/or informative enough when computed with reasonable runtimes. We underline also the fact that the parameters of the algorithms have *not* been tuned individually for each instance: the parametric choices described in the foregoing sections have been run without modification on all the large instances.

## 11. Conclusions

In this paper, we have defined and solved a bi-objective dock assignment problem with trailer transportation, with a clear hierarchy between the two objectives. We have explored the limits of the instance sizes that can be solved to guaranteed optimality within acceptable running times by means of integer programming and branch and bound. It turns out that these limits are too low to be of any use in the practical case that was the prime motivation for

Table 11: Final computational results for the large instances. An asterisk '*' in the fourth column indicates that the length of the time horizon was extended to $H_{\max} = 144$.

| $m$ | $n$ | $\tau$ | | recovery beam | | TS swap | | hybrid seq | | hybrid alt | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ | $z_1$ | $z_2(z_1)$ |
| 36 | 216 | 4 | | 555 | 19956 | 531 | 19731 | 513 | 19715 | 511 | 19701 |
| 36 | 216 | 5 | | 177 | 18458 | 157 | 18082 | 159 | 18114 | 161 | 18057 |
| 36 | 288 | 5 | | 549 | 29486 | 536 | 28488 | 522 | 28388 | 501 | 28219 |
| 36 | 288 | 6 | | 493 | 28848 | 416 | 28107 | 430 | 27945 | 421 | 27888 |
| 36 | 360 | 6 | * | 960 | 45951 | 736 | 44587 | 732 | 44153 | 728 | 44094 |
| 36 | 360 | 7 | * | 739 | 44400 | 654 | 43772 | 664 | 43496 | 667 | 43528 |
| 40 | 240 | 4 | | 846 | 22037 | 831 | 21504 | 823 | 21486 | 821 | 21440 |
| 40 | 240 | 5 | | 368 | 19318 | 342 | 19408 | 344 | 19166 | 343 | 19053 |
| 40 | 320 | 5 | | 600 | 34333 | 573 | 32859 | 558 | 32888 | 583 | 32745 |
| 40 | 320 | 6 | | 305 | 33216 | 275 | 32158 | 250 | 31953 | 239 | 31781 |
| 40 | 400 | 6 | * | 1075 | 49433 | 884 | 48417 | 811 | 48092 | 806 | 47837 |
| 40 | 400 | 7 | * | 895 | 48541 | 676 | 47300 | 673 | 47251 | 670 | 47108 |
| 44 | 264 | 5 | | 654 | 23608 | 592 | 22640 | 590 | 22773 | 598 | 22764 |
| 44 | 264 | 6 | | 330 | 22145 | 265 | 21658 | 272 | 21300 | 280 | 21156 |
| 44 | 352 | 6 | | 578 | 36087 | 508 | 35074 | 517 | 35080 | 510 | 34944 |
| 44 | 352 | 7 | | 457 | 35436 | 409 | 34550 | 408 | 34576 | 408 | 34550 |
| 44 | 440 | 7 | * | 1256 | 53177 | 1052 | 51937 | 1044 | 51749 | 1058 | 51910 |
| 44 | 440 | 8 | * | 1130 | 52423 | 986 | 51455 | 980 | 51343 | 982 | 51358 |
| 48 | 288 | 5 | | 887 | 26904 | 857 | 26294 | 834 | 26416 | 838 | 26327 |
| 48 | 288 | 6 | | 435 | 24407 | 387 | 24093 | 386 | 23884 | 383 | 23824 |
| 48 | 384 | 6 | | 1389 | 41085 | 1447 | 39885 | 1391 | 39141 | 1563 | 38634 |
| 48 | 384 | 7 | | 961 | 39538 | 936 | 38201 | 916 | 37942 | 928 | 37834 |
| 48 | 480 | 7 | * | 1477 | 60813 | 1365 | 59588 | 1296 | 59330 | 1340 | 59368 |
| 48 | 480 | 8 | * | 1374 | 59911 | 1199 | 59028 | 1198 | 59139 | 1194 | 58938 |
| average | | | | 770 | | 692 | | 680 | | 689 | |
| gap | | | | 21.01% | | 7.86% | | 6.4% | | 7.09% | |

undertaking this work, which can require the planning of up to 48 gates, 480 trailers and over 1000 operations. We have therefore also examined the performance of different heuristics for solving large instances, namely a truncated branch-and-bound algorithm, a standard and a recovery beam search algorithm, and a tabu search algorithm. We apply Lagrangian relaxation for computing lower bounds, which can also be transformed into upper bounds (candidate solutions) via a schedule generation scheme. With respect to beam search, a recovery phase via a pseudo-dominance rule and a limited neighborhood search improves the standard implementation. This improvement is not as substantial as reported in literature for other problems, however. We find that a hybrid implementation of tabu search with recovery beam search produces the best results, generating high-quality solutions to realistic instances within reasonable computation times.

Overall, with the current size and complexity of the search space, we conclude that a strong randomization is better than a structured search of the solution space by means of, e.g., beam search alone, presumably because too much overhead runtime is incurred in steering the procedure in the latter case. The main problem with branch and bound and beam search resides in the computation time for the lower bounds. Different bounds have been implemented, but all are either computationally too expensive or not tight enough to be beneficial. As a result, the best-performing implementations explore more nodes by not computing any lower bounds. As a first consequence, nodes in the search tree cannot be pruned. Secondly, lower bounds are also not available to guide the choice of the next branching alternative. For this reason, a challenging but valuable direction for future research is the development of efficient lower bounds. These may be interesting even if they are not tight, as long as they represent rather well the quality of a node.

We have proposed to use a baseline schedule as a general guideline for the operations, which is gradually adjusted to reflect more accurate information as it becomes available, and not as a strict prescriptive plan providing a minute-to-minute description of the timing of the activities throughout the day. There is a large body of scientific literature on scheduling under uncertainty that confirms that when uncertainty is not too pervasive, then developing an indicative static baseline schedule and updating it as time progresses, allows to achieve a better quality for the resulting schedule than when no such baseline schedule is used and tasks are simply handled on a first-come-first-served basis (see for instance Vieira et al. (2003) or Herroelen and Leus (2005)). For future work, a thorough simulation exercise might be set up to verify whether adhering to such an iteratively updated baseline schedule is more valuable than the use of simple real-time dispatching rules also in this particular scheduling environment.

## References


M. Azizoğlu and O. Kirca. Tardiness minimization on parallel machines. *International Journal of Production Economics*, 55:163–168, 1998.

M. Azizoğlu and O. Kirca. On the minimization of total weighted flow time with identical and uniform parallel machines. *European Journal of Operational Research*, 113:91–100, 1999.

K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

K. R. Baker and J. C. Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6:7–16, 2003.

M. O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16:21–38, 2011.

L. Berghman, R. Leus, and F. C. R. Spieksma. Optimal solutions for a dock assignment problem with trailer transportation. *Annals of Operations Research*, 213(1):3–25, 2014.

R. Bisiani. Beam search. In *Encyclopedia of Artificial Intelligence*, pages 1467–1568. Wiley Interscience Publication, 1992.

J. Blazewicz, P. Dell'Olmo, and M. Drozdowski. Scheduling of client-server applications. *International Transactions in Operational Research*, 6:345–363, 1999.

N. Boysen and M. Fliedner. Cross dock scheduling: Classification, literature review and research agenda. *Omega*, 38:413–422, 2010.

J. S. Chen, J. C. H. Pan, and C. K. Wu. Minimizing makespan in reentrant flow-shops using hybrid tabu search. *The International Journal of Advanced Manufacturing Technology*, 34:353–361, 2007.

S. W. Choi and Y. D. Kim. Minimizing makespan on an m-machine re-entrant flowshop. *Computers & Operations Research*, 35:1684–1696, 2008.

F. Della Croce and V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53:1275–1280, 2002.

F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10:89–104, 2004.

E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38(12):1803–1818, 1992.

A. Drexl. Scheduling of project networks by job assignment. *Management Science*, 37(12): 1590–1606, 1991.

B. Esteve, C. Aubijoux, A. Chartier, and V. T'Kindt. A recovering beam search algorithm for the single machine Just-in-Time scheduling problem. *European Journal of Operational Research*, 172:798–813, 2006.

M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

B. Gacias, C. Artigues, and P. Lopez. Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, 37:2141–2151, 2010.

M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: motivation, examples, and implications. *Journal of the Association for Computing Machinery*, 25(3):499–508, 1978.

M. Ghirardi and C. N. Potts. Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. *European Jounal of Operational Research*, 165:457–467, 2005.

F. Glover. Tabu search – part 1. *ORSA Journal on Computing*, 1(3):190–206, 1989.

S. C. Graves, H. C. Meal, D. Stefeka, and A. H. Zeghmi. Scheduling of re-entrant flow shops. *Journal of Operations Management*, 3(4):197–207, 1983.

J. N. C. Gupta, K. Krüger, V. Lauff, F. Werner, and Y. N. Sotskov. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers & Operations Research*, 29:1417–1439, 2002.

M. Haouari, L. Hidri, and A. Gharbi. Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64:107–124, 2006.

W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167: 592–623, 2005.

ILOG. *ILOG CPLEX 11.0 User's Manual*. ILOG Inc., 2008. Available online at `http://www.decf.berkeley.edu/help/apps/ampl/cplex-doc/`.

T. Kis and E. Pesch. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. *European Journal of Operational Research*, 164:592–608, 2005.

R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(11):1693–1703, 1995.

W. Kubiak, S. X. C. Lou, and Y. Wang. Mean flow time minimization in re-entrant job-shops with a hub. *Operations Research*, 44:764–776, 1996.

R. Linn and W. Zhang. Hybrid flow shop scheduling: A survey. *Computers and Industrial Engineering*, 37:57–61, 1999.

Z. Miao, A. Lim, and H. Ma. Truck dock assignment problem with operational time constraint within crossdocks. *European Journal of Operational Research*, 192:105–115, 2009.

R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Resource constrained project scheduling: Computing lower bounds by solving minimum cut problems. *Lecture Notes in Computer Science*, 1643:139–150, 1999.

R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

O. Moursli and Y. Pochet. A branch-and-bound algorithm for the hybrid flowshop. *International Journal of Production Economics*, 64:113–125, 2000.

R. Nessah, F. Yalaoui, and C. Chu. A branch-and-bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. *Computers & Operations Research*, 35:1176–1190, 2008.

T. Nichi, Y. Hiranaka, and M. Inuiguchi. Lagrangian relaxation with cut generation for hybrid flowshop scheduling problems to minimize the total weighted tardiness. *Computers & Operations Research*, 37:189–198, 2010.

C. D. Paternina-Arboleda, J. R. Montoya-Torres, M. J. Acero-Domingues, and M. C. Herrera-Hernandez. Scheduling jobs on a k-stage flexible flow-shop. *Annals of Operations Research*, 164:29–40, 2008.

M. L. Pinedo. *Scheduling. Theory, Algorithms, and Systems.* Springer, third edition, 2008.

R. M. V. Rachamadugu and T. E. Morton. Myopic heuristics for the weighted tardiness problem on identical parallel machines. Working Paper #28-81-82, Graduate School of Industrial Administration, Carnegie-Mellon University, 1981.

I. Ribas, R. Leisten, and J. M. Framiñan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37:1439–1454, 2010.

R. Sadykov and L. A. Wolsey. Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS Journal on Computing*, 18(2):209–217, 2006.

S. C. Sarin and R. Hariharan. A two machine bicriteria scheduling problem. *International Journal of Production Economics*, 65:125–139, 2000.

S. O. Shim and Y. D. Kim. Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research*, 177:135–146, 2007.

L. Tang and H. Xuan. Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. *Journal of the Operational Research Society*, 57:316–324, 2006.

V. T'kindt, J. N. D. Gupta, and J. C. Billaut. Two-machine flowshop scheduling with a secondary criterion. *Computers & Operations Research*, 30:505–526, 2003.

J. M. S. Valente and R. A. F. S. Alves. Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers & Industrial Engineering*, 48:363–375, 2005.

J. van Belle, P. Valckenaers, and D. Cattrysse. Cross docking: State of the art. *Omega*, 40 (8):827–846, 2012.

G.E. Vieira, J.W. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

A. Vignier, J. C. Billaut, and C. Proust. Hybrid flowshop scheduling problems: State of the art. *RAIRO Operations Research*, 33(2):117–183, 1999.

W. Yu and P. J. Egbelu. Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. *International Journal of Production Economics*, 184: 377–396, 2008.